Assignment 3 - Part A

Generated by Doxygen 1.12.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 BinControl Class Reference

A class to represent a bin in the persistent hash map.

```
#include <persistent_hashmap.h>
```

Public Member Functions

- BinControl (int bin_id, std::string bin_path, int cache_capacity)

    Constructs a new BinControl object.
- void insert (std::string key, std::string value)

    Inserts a key-value pair into the bin.
- bool erase (std::string key)

    Erases a key-value pair from the bin.
- std::optional< std::string > get (std::string key)

    Retrieves the value of a key from the bin.
- void compressFile ()

    Compresses the bin file.
- void binCheck ()

    Checks and fixes the bin file.

3.1.1 Detailed Description

A class to represent a bin in the persistent hash map.

This class provides a class to represent a bin in the persistent hash map. A bin is a file that stores key-value pairs. The class provides methods to insert, erase, and get key-value pairs from the bin. It also provides methods to compress the file and check/fix the file.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BinControl()

```
BinControl::BinControl (
            int bin_id,
            std::string bin_path,
            int cache_capacity)
```

Constructs a new BinControl object.

Parameters

bin_id	The ID of the bin.
bin_path	The path of the bin.
cache_capacity	The capacity of the LRU cache segment.

3.1.3 Member Function Documentation

3.1.3.1 binCheck()

```
void BinControl::binCheck ()
```

Checks and fixes the bin file.

Checks the bin file for consistency. Reads entries and checks if they are valid. Invalid entries, which may occur towards the end of the file, are removed.

This method checks the bin file for any inconsistencies and fixes them if found. Writer lock to write to the file, allowing only one writer at a time. Ensures no other thread accesses the file while it is being checked.

Iterating through the file to read each entry and check if it is valid

Checking if the entry fits within the file size

3.1.3.2 compressFile()

```
void BinControl::compressFile ()
```

Compresses the bin file.

This method compresses the bin file by removing any deleted key-value pairs and reorganizing the file accordingly. Writer lock to write to the file, allowing only one writer at a time. Ensures no other thread accesses the file while it is being compressed.

< The current read position in the file

< The current write position in the file

Iterating through the file to read each entry, and writing it back to the file if it is not marked as deleted

If the entry is marked as deleted, skip it

Writing the non-deleted entry back to the file

Updating the write position to the end of the written entry

Truncating the file to the final write position

3.1.3.3 erase()

```
bool BinControl::erase (
             std::string key)
```

Erases a key-value pair from the bin.

Parameters

key	The key of the key-value pair to erase.

Returns

True if the key-value pair was erased, false otherwise.

Writer lock to write to the cache and the file, allowing only one writer at a time

Remove the key from the cache

Iterating through the file to find the key, and if found, mark it as deleted and return true

Skip this entry if it is marked as deleted or if lengths mismatch

If the key is found, mark it as deleted and return true

3.1.3.4 get()

```
std::optional< std::string > BinControl::get (
            std::string key)
```

Retrieves the value of a key from the bin.

Parameters

key	The key to retrieve the value for.

Returns

The value of the key, if it exists.

Reader lock to read from the cache and the file, allowing multiple readers at a time

If the key is found in the cache, return the value

Iterating through the file to find the key, and if found, updating the cache and returning the value

Skip this entry if it is marked as deleted or if lengths mismatch

If the key is found, read the value, update the cache, and return the value

3.1.3.5 insert()

```
void BinControl::insert (
            std::string key,
            std::string value)
```

Inserts a key-value pair into the bin.

Parameters

key	The key of the key-value pair.
value	The value of the key-value pair.

Ensure that only one thread can write to the bin at a time. This includes writing to the cache and the file.

Iterating through the file to find the key if it exists, and mark it as deleted if it does

Storing the starting position of the current entry, to be used for marking the entry as deleted

Skip this entry if it is marked as deleted or if lengths mismatch, as it is not the key we are looking for

If the key is found, mark it as deleted

We don't need to read the value, so we skip it

Write the new key-value pair to the end of the file

The documentation for this class was generated from the following files:

- include/persistent_hashmap.h
- src/persistent_hashmap.cpp

3.2 LRUCacheElement Class Reference

A class to represent an element in the LRU cache.

```
#include <lru_cache.h>
```

Public Member Functions

- LRUCacheElement (std::string key, std::string value)

  Constructs a new LRUCacheElement object.

Public Attributes

- std::string **key** {}

  The key of the element.
- std::string **value** {}

  The value of the element.

3.2.1 Detailed Description

A class to represent an element in the LRU cache.

This class provides a class to represent an element in the LRU cache.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 LRUCacheElement()

```
LRUCacheElement::LRUCacheElement (
            std::string key,
            std::string value)
```

Constructs a new LRUCacheElement object.

Parameters

key	The key of the element.
value	The value of the element.

The documentation for this class was generated from the following files:

- include/lru_cache.h
- src/lru_cache.cpp

3.3 LRUCacheSegment Class Reference

A class to represent a segment in the LRU cache.

```
#include <lru_cache.h>
```

Public Member Functions

- LRUCacheSegment (int capacity)

  Constructs a new LRUCacheSegment object.
- void put (std::string key, std::string value)

  Inserts a new element into the LRU cache segment.
- std::optional< std::string > get (std::string key)

  Retrieves the value of an element from the LRU cache segment.
- void remove (std::string key)

  Removes an element from the LRU cache segment.

3.3.1 Detailed Description

A class to represent a segment in the LRU cache.

This class provides a class to represent a segment in the LRU cache.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 LRUCacheSegment()

```
LRUCacheSegment::LRUCacheSegment (
            int capacity)
```

Constructs a new LRUCacheSegment object.

Parameters

capacity	The capacity of the LRU cache segment.

3.3.3 Member Function Documentation

3.3.3.1 get()

```
std::optional< std::string > LRUCacheSegment::get (
            std::string key)
```

Retrieves the value of an element from the LRU cache segment.

Parameters

key	The key of the element.

Returns

The value of the element, if it exists.

Ensure that only one thread can access the cache segment at a time

Check if the key exists in the cache segment. If it does, move the element to the front of the list. Otherwise, return an empty optional.

3.3.3.2 put()

```
void LRUCacheSegment::put (
            std::string key,
            std::string value)
```

Inserts a new element into the LRU cache segment.

Parameters

key	The key of the element.
value	The value of the element.

Ensure that only one thread can access the cache segment at a time

Check if the key already exists in the cache segment. If it does, update the value and move the element to the front of the list. Otherwise, insert a new element at the front of the list and evict the least recently used element if the capacity is exceeded.

Update existing element

Insert new element

Evict least recently used element if capacity is exceeded

3.3.3.3 remove()

```
void LRUCacheSegment::remove (
            std::string key)
```

Removes an element from the LRU cache segment.

Parameters

key	The key of the element.

Ensure that only one thread can access the cache segment at a time

Check if the key exists in the cache segment. If it does, remove the element from the list and map.

The documentation for this class was generated from the following files:

- include/lru_cache.h
- src/lru_cache.cpp

3.4 PersistentHashMap Class Reference

A class to represent a disk-persistent hash map.

```
#include <persistent_hashmap.h>
```

Public Types

- enum class INITIALIZATION_MODE { CREATE = 0 , OPEN = 1 }

  An enumeration to specify the initialization mode of the persistent hash map.

Public Member Functions

- PersistentHashMap (std::string directory, int num_bins, INITIALIZATION_MODE mode=INITIALIZATION_MODE::CREATE)

  Constructs a new PersistentHashMap object.
- ∼PersistentHashMap ()

  Destroys the PersistentHashMap object.
- void insert (std::string key, std::string value)

  Inserts a key-value pair into the hash map.
- bool erase (std::string key)

  Erases a key-value pair from the hash map.
- std::optional< std::string > get (std::string key)

  Retrieves the value of a key from the hash map.

3.4.1 Detailed Description

A class to represent a disk-persistent hash map.

This class provides a class to represent a persistent hash map. The persistent hash map stores key-value pairs in bins, which are files that store the key-value pairs. The class provides methods to insert, erase, and get key-value pairs from the hash map. It also provides a garbage collector to compress the bins.

3.4.2 Member Enumeration Documentation

3.4.2.1 INITIALIZATION_MODE

```
enum class PersistentHashMap::INITIALIZATION_MODE  [strong]
```

An enumeration to specify the initialization mode of the persistent hash map.

This enumeration specifies the initialization mode of the persistent hash map. The persistent hash map can be created or opened.

Enumerator

CREATE	Create a new persistent hash map.
OPEN	Opens an existing persistent hash map.

3.4.3 Constructor & Destructor Documentation

3.4.3.1 PersistentHashMap()

```
PersistentHashMap::PersistentHashMap (
            std::string directory,
            int num_bins,
            INITIALIZATION_MODE mode = INITIALIZATION_MODE::CREATE)
```

Constructs a new PersistentHashMap object.

Parameters

directory	The directory where the bins are stored.
num_bins	The number of bins in the hash map.
mode	The initialization mode of the hash map.

Creating bin controls for each bin

If the mode is CREATE, clear the directory and create new bucket files. Overwrite existing files if they exist.

If the mode is OPEN, create new bucket files if they do not exist. Otherwise, check the existing files for consistency.

Start the garbage collector thread

3.4.3.2 ∼PersistentHashMap()

```
PersistentHashMap::∼PersistentHashMap ()
```

Destroys the PersistentHashMap object.

Stop the garbage collector thread

3.4.4 Member Function Documentation

3.4.4.1 erase()

```
bool PersistentHashMap::erase (
             std::string key)
```

Erases a key-value pair from the hash map.

Parameters

key	The key of the key-value pair to erase.

Returns

True if the key-value pair was erased, false otherwise.

3.4.4.2 get()

```
std::optional< std::string > PersistentHashMap::get (
             std::string key)
```

Retrieves the value of a key from the hash map.

Parameters

key	The key to retrieve the value for.

Returns

The value of the key, if it exists.

3.4.4.3 insert()

```
void PersistentHashMap::insert (
            std::string key,
            std::string value)
```

Inserts a key-value pair into the hash map.

Parameters

key	The key of the key-value pair.
value	The value of the key-value pair.

The documentation for this class was generated from the following files:

- include/persistent_hashmap.h
- src/persistent_hashmap.cpp

3.5 SpinLock Class Reference

A simple spin lock implementation using atomic_flag.

```
#include <lru_cache.h>
```

Public Member Functions

- void **lock** ()
  Locks the spin lock.
- void **unlock** ()
  Unlocks the spin lock.

3.5.1 Detailed Description

A simple spin lock implementation using atomic_flag.

This class provides a simple spin lock implementation using atomic_flag.

The documentation for this class was generated from the following files:

- include/lru_cache.h
- src/lru_cache.cpp

3.6 StorageEngine Class Reference

A class to represent a disk-persistent key-value store.

```
#include <storage_engine.h>
```

Public Types

- enum class INITIALIZATION_MODE { CREATE = 0 , OPEN = 1 }

  An enumeration to specify the initialization mode of the database.

Public Member Functions

- StorageEngine (std::string directory, int num_bins, INITIALIZATION_MODE mode=INITIALIZATION_MODE::CREATE)

  Constructs a new StorageEngine object.
- ∼**StorageEngine** ()

  Destroys the StorageEngine object.
- void insert (std::string key, std::string value)

  Inserts a key-value pair into the key-value store.
- bool erase (std::string key)

  Erases a key-value pair from the key-value store.
- std::optional< std::string > get (std::string key)

  Retrieves the value of a key from the key-value store.

3.6.1 Detailed Description

A class to represent a disk-persistent key-value store.

This class provides a class to represent a disk-persistent key-value store. The key-value store stores key-value pairs in a persistent hash map. The class provides an interface to insert, erase, and get key-value pairs from the key-value store.

3.6.2 Member Enumeration Documentation

3.6.2.1 INITIALIZATION_MODE

```
enum class StorageEngine::INITIALIZATION_MODE  [strong]
```

An enumeration to specify the initialization mode of the database.

This enumeration specifies the initialization mode of the database. The database can be created or opened.

Enumerator

CREATE	Create a new database.
OPEN	Opens an existing database.

3.6.3 Constructor & Destructor Documentation

3.6.3.1 StorageEngine()

```
StorageEngine::StorageEngine (
            std::string directory,
            int num_bins,
            INITIALIZATION_MODE mode = INITIALIZATION_MODE::CREATE)
```

Constructs a new StorageEngine object.

Parameters

directory	The directory where the key-value pairs are stored.
num_bins	The number of bins in the hash map.
mode	The initialization mode of the database.

3.6.4 Member Function Documentation

3.6.4.1 erase()

```
bool StorageEngine::erase (
            std::string key)
```

Erases a key-value pair from the key-value store.

Parameters

key	The key of the key-value pair to erase.

Returns

True if the key-value pair was erased, false otherwise.

3.6.4.2 get()

```
std::optional< std::string > StorageEngine::get (
            std::string key)
```

Retrieves the value of a key from the key-value store.

Parameters

key	The key to retrieve the value for.

Returns

The value of the key, if it exists.

3.6.4.3 insert()

```
void StorageEngine::insert (
            std::string key,
            std::string value)
```

Inserts a key-value pair into the key-value store.

Parameters

key	The key of the key-value pair.
value	The value of the key-value pair.

The documentation for this class was generated from the following files:

- include/storage_engine.h
- src/storage_engine.cpp

Chapter 4

File Documentation

4.1 lru_cache.h

```
#pragma once
#include <list>
#include <unordered_map>
#include <string>
#include <atomic>
#include <memory>
#include <optional>

class SpinLock {
    std::atomic_flag flag {ATOMIC_FLAG_INIT};

public:
    void lock();

    void unlock();
};

struct LRUCacheElement {
    std::string key {};

    std::string value {};

    LRUCacheElement(std::string key, std::string value);
};

class LRUCacheSegment {
    std::list<std::shared_ptr<LRUCacheElement» lruList {};

    std::unordered_map<std::string, std::shared_ptr<LRUCacheElement» lruMap {};

    int capacity {};

    SpinLock lock {};

public:
    LRUCacheSegment(int capacity);

    void put(std::string key, std::string value);

    std::optional<std::string> get(std::string key);

    void remove(std::string key);
};
```

4.2 persistent_hashmap.h

```
#pragma once
#include <lru_cache.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <shared_mutex>
#include <thread>

```

```
#ifdef LRU_CACHE_SIZE
int const LRU_CACHE_CAPACITY = LRU_CACHE_SIZE;
#else
int const LRU_CACHE_CAPACITY = 64;
#endif

#ifdef GC_INTERVAL
int const GARBAGE_COLLECTOR_INTERVAL = GC_INTERVAL;
#else
int const GARBAGE_COLLECTOR_INTERVAL = 30;
#endif

class BinControl {
    int bin_id {};

    std::string bin_path {};

    std::shared_mutex mutex {};

    LRUCacheSegment cache {LRU_CACHE_CAPACITY};

public:
    BinControl(int bin_id, std::string bin_path, int cache_capacity);

    void insert(std::string key, std::string value);

    bool erase(std::string key);

    std::optional<std::string> get(std::string key);

    void compressFile();

    void binCheck();
};

class PersistentHashMap {
    std::string directory {};

    int num_bins {};

    std::vector<std::unique_ptr<BinControl» bin_controls {};

    std::thread gc_thread {};

    std::atomic<bool> stop_gc {false};

    void runGarbageCollector();

    int hash(std::string key) const;

public:
    enum class INITIALIZATION_MODE {
        CREATE = 0,

        OPEN = 1
    };

    PersistentHashMap(std::string directory, int num_bins, INITIALIZATION_MODE mode =
INITIALIZATION_MODE::CREATE);

    ~PersistentHashMap();

    void insert(std::string key, std::string value);

    bool erase(std::string key);

    std::optional<std::string> get(std::string key);
};
```

4.3 storage_engine.h

```
#pragma once
#include <string>
#include <memory>
#include <optional>

class PersistentHashMap;

class StorageEngine {
public:
    enum class INITIALIZATION_MODE {
        CREATE = 0,

        OPEN = 1
    };

    StorageEngine(std::string directory, int num_bins, INITIALIZATION_MODE mode =
INITIALIZATION_MODE::CREATE);

    ~StorageEngine();

    void insert(std::string key, std::string value);

    bool erase(std::string key);

    std::optional<std::string> get(std::string key);

private:
    std::unique_ptr<PersistentHashMap> pImpl;
};
```

Index